

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-268074

(43)Date of publication of application : 29.09.2000

(51)Int.Cl.

G06F 17/50

G01R 31/28

G06F 9/38

(21)Application number : 11-074118

(71)Applicant : TOSHIBA CORP

(22)Date of filing : 18.03.1999

(72)Inventor : KONO KAZUYOSHI

IMAI HIROSHI

MIZUNO ATSUSHI

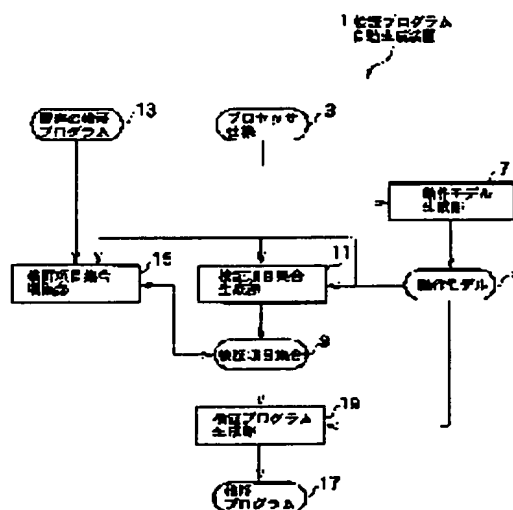
KAMIYA HIRONORI

(54) DEVICE AND METHOD FOR AUTOMATICALLY GENERATING VERIFICATION PROGRAM AND DEVICE AND METHOD FOR AUTOMATICALLY GENERATING PROPERTY

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a device and a method for automatically generating a verification program and a property for verifying the design (operation) of a processor.

SOLUTION: An automatic verification program generator 1 is composed of an operation model generating part 7 for inputting a processor specification 3 and generating an operation model 5 of the processor, a verification item set generating part 11 for inputting the operation model 5 and the processor specification 3 and generating a verification item set, a verification item set editing part 15 for inputting an existent verification program 13 and the operation model 5 and editing a verification item set 9 and a verification program generating part 19 for inputting the verification item set 9 and the operation model 5 and generating a verification program 17 corresponding to individual verification items.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's
decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2000 Japanese Patent Office

(19)日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11)特許出願公開番号
特開2000-268074
(P2000-268074A)

(43)公開日 平成12年9月29日(2000.9.29)

(51)Int.Cl. ⁷	識別記号	F I	テーマコード(参考)
G 0 6 F 17/50		G 0 6 F 15/60	6 7 0 K 2 G 0 3 2
G 0 1 R 31/28		9/38	3 5 0 A 5 B 0 1 3
G 0 6 F 9/38	3 5 0	G 0 1 R 31/28	F 5 B 0 4 6 9 A 0 0 1

審査請求 未請求 請求項の数4 O L (全 13 頁)

(21)出願番号 特願平11-74118

(22)出願日 平成11年3月18日(1999.3.18)

(71)出願人 000003078

株式会社東芝

神奈川県川崎市幸区堀川町72番地

(72)発明者 河野 和義

神奈川県川崎市幸区堀川町580番1号 株

式会社東芝半導体システム技術センター内

(72)発明者 今井 浩史

神奈川県川崎市幸区堀川町580番1号 株

式会社東芝半導体システム技術センター内

(74)代理人 100083806

弁理士 三好 秀和 (外7名)

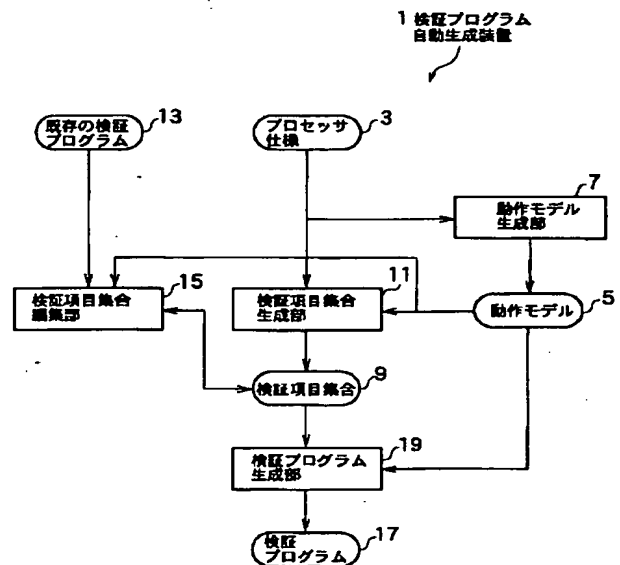
最終頁に続く

(54)【発明の名称】 検証プログラム自動生成装置および方法並びにプロパティ自動生成装置および方法

(57)【要約】

【課題】 プロセッサの設計検証（動作検証）を行なうための検証プログラムおよびプロパティを自動生成する装置および方法を提供する。

【解決手段】 プロセッサ仕様3を入力し、プロセッサの動作モデル5を生成する動作モデル生成部7と、動作モデル5およびプロセッサ仕様3を入力し、検証項目集合を生成する検証項目集合生成部11と、既存の検証プログラム13および動作モデル5を入力し、検証項目集合9を編集する検証項目集合編集部15と、検証項目集合9および動作モデル5を入力し、各検証項目に対応する検証プログラム17を生成する検証プログラム生成部19とから構成される検証プログラム自動生成装置1である。



【特許請求の範囲】

【請求項1】 プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成部と、前記動作モデルおよび前記プロセッサ仕様を入力し、検証項目集合を生成する検証項目集合生成部と、既存の検証プログラムおよび前記動作モデルを入力し、前記検証項目集合を編集する検証項目集合編集部と、前記検証項目および前記動作モデルを入力し、各検証項目に対応する検証プログラムを生成する検証プログラム生成部とを少なくとも有することを特徴とする検証プログラム自動生成装置。

【請求項2】 プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成ステップと、前記動作モデルおよび前記プロセッサ仕様を入力し、検証項目集合を生成する検証項目集合生成ステップと、既存の検証プログラムおよび前記動作モデルを入力し、前記検証項目集合を編集する検証項目集合編集ステップと、前記検証項目および前記動作モデルを入力し、各検証項目に対応する検証プログラムを生成する検証プログラム生成ステップとを少なくとも有することを特徴とする検証プログラム自動生成方法。

【請求項3】 プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成部と、その動作モデルおよびプロセッサの回路記述を入力し、前記プロセッサのプロパティを生成するプロパティ生成部とを少なくとも有することを特徴とするプロパティ自動生成装置。

【請求項4】 プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成ステップと、その動作モデルおよびプロセッサの回路記述を入力し、前記プロセッサのプロパティを生成するプロパティ生成ステップとを少なくとも有することを特徴とするプロパティ自動生成方法。

【発明の詳細な説明】**【0001】**

【発明の属する技術分野】 本発明は、プロセッサの設計検証を行なうための検証プログラムを自動生成する装置および方法に関する。また、本発明は、プロセッサの動作検証を行なうためのプロパティを自動生成する装置および方法に関する。

【0002】

【従来の技術】 高効率な検証プログラムの生成はプロセッサの設計検証容易化のためには不可欠である。しかし、検証プログラムの検証項目作成は人手によるため、検証項目の漏れ、誤った検証項目の作成などが起こり得る。さらに、プロセッサの論理が複雑になると検証項目の個数も膨大となり、人手での検証項目の作成はより一層困難となる。また、すでに作成済の検証プログラムが存在すれば、その検証プログラムの検証項目に関しては

新たに検証プログラムを作成する必要はない。ところが、既存の検証プログラムが目的の検証項目を満たすものであるか否かは実際にその検証プログラムをプロセッサに与えてシミュレーションしてみなければわからない。したがって、検証プログラムの個数が多い場合もしくは検証プログラムのサイズが大きい場合には膨大なシミュレーション時間を要する。

【0003】 また、検証プログラムはプロセッサの動作モデルに基づいて作成されるが、その動作モデルの作成は設計者がプロセッサの仕様を参考に行なっている。したがって、動作モデル作成にはかなりの時間が必要であり、また、プロセッサの仕様が変更されればその仕様を満たすように動作モデルを再び作成し直す必要があった。

【0004】 一方、製品サイクルが短くなり、設計から市場投入までの時間の短縮が求められている。プロセッサの設計では設計中の論理回路の動作検証に多くの時間が費やされており、この時間短縮のため動作検証にシミュレーションを用いない形式的検証方法が採用されてきている。

【0005】 形式的検証方法のうちモデル検証と呼ばれる方法は設計対象のプロセッサの動作をプロパティとして記述し、プロセッサがプロパティを満足するかどうかを形式的に検証する。このようにして、設計対象の論理回路が仕様通りの動作を行うかどうかを検証する。

【0006】 従来、プロパティは設計者が仕様と回路記述を見ながら作成していたが、プロセッサの大規模化に伴いその仕様が複雑化し、プロパティの作成に要する時間が長期化する傾向にある。また、人手によるため、正しいプロパティを作成することが難しいという問題がある。

【0007】

【発明が解決しようとする課題】 本発明は、このような課題を解決し、プロセッサの設計検証を行なうための検証プログラムを自動生成する装置および方法を提供する。また、本発明は、プロセッサの動作検証を行なうためのプロパティを自動生成する装置および方法を提供する。

【0008】

【課題を解決するための手段】 上記課題を解決するため、本発明の第1の特徴は、プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成部と、前記動作モデルおよび前記プロセッサ仕様を入力し、検証項目集合を生成する検証項目集合生成部と、既存の検証プログラムおよび前記動作モデルを入力し、前記検証項目集合を編集する検証項目集合編集部と、前記検証項目および前記動作モデルを入力し、各検証項目に対応する検証プログラムを生成する検証プログラム生成部とを少なくとも有する検証プログラム自動生成装置であることである。

【0009】本発明の第2の特徴は、プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成ステップと、前記動作モデルおよび前記プロセッサ仕様を入力し、検証項目集合を生成する検証項目集合生成ステップと、既存の検証プログラムおよび前記動作モデルを入力し、前記検証項目集合を編集する検証項目集合編集ステップと、前記検証項目および前記動作モデルを入力し、各検証項目に対応する検証プログラムを生成する検証プログラム生成ステップとを少なくとも有する検証プログラム自動生成方法であることである。

【0010】本発明の第3の特徴は、プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成部と、その動作モデルおよびプロセッサの回路記述を入力し、前記プロセッサのプロパティを生成するプロパティ生成部とを少なくとも有するプロパティ自動生成装置であることである。

【0011】本発明の第4の特徴は、プロセッサ仕様を入力し、プロセッサの動作モデルを生成する動作モデル生成ステップと、その動作モデルおよびプロセッサの回路記述を入力し、前記プロセッサのプロパティを生成するプロパティ生成ステップとを少なくとも有するプロパティ自動生成方法であることである。

【0012】

【発明の実施の形態】（第1の実施の形態）図1は本発明の第1の実施の形態に係る検証プログラム自動生成装置の構成を示すブロック図である。本発明の第1の実施の形態は、パイプライン化したプロセッサを設計する場合に本発明を適用した例である。図1に示すように、本発明の第1の実施の形態に係る検証プログラム自動生成装置1は、プロセッサの仕様3を入力し、プロセッサの動作モデル5を生成する動作モデル生成部7と、プロセッサの仕様3および動作モデル5を入力し、検証項目集合9を生成する検証項目集合生成部11と、既存の検証プログラム13および動作モデル5を入力し、検証項目集合9を編集する検証項目集合編集部15と、動作モデル5および検証項目集合9を入力し、各検証項目に対応する検証プログラム17を生成する検証プログラム生成部19とを備えて構成される。

【0013】プロセッサの仕様3には設計対象のプロセッサの命令セット・アーキテクチャ（ISA: instruction set architecture）仕様、各命令のパイプライン動作仕様、パイプラインのストール（stall）規則が含まれる。図2に命令セット・アーキテクチャ仕様の例を示す。図2に示した例では、命令セット・アーキテクチャ仕様として6個の命令およびその動作が与えられている。「nop」命令は何も行なわない命令、「add」命令は整数の加算を行なう命令、「sub」命令は整数の減算を行なう命令、「fadd」命令は浮動小数点数の加算を行なう命令、「fsub」命令は浮動少数点数の減算を行なう命令、「ld」命令はメモリからデータ

をロードする命令である。

【0014】図3にパイプライン動作仕様の例を示す。パイプライン制御の動作を考えた場合に全く同様に動作する命令はグループ化することができるようにしておく。図3に示した例では、「add」命令および「sub」命令を「int」命令に、「fadd」命令および「fsub」命令を「fpu」命令にグループ化する。各パイプライン・ステージにおける動作（パイプライン動作）は次の通りである。「FET」ステージは命令のフェッチ、デコードおよびレジスタ・フェッチを行なう。「ALU」ステージは整数演算の実行を行なう。「FPU」パイプライン・ステージは浮動小数点演算の実行を行なう。「MEM」ステージはメモリ・アクセスを行なう。「WB」ステージはレジスタへの書き込みを行なう。

【0015】パイプラインのストール規則は構造ハザード・ストール規則とデータ・ハザード・ストール規則からなる。構造ハザード・ストール規則はパイプライン動作中に構造ハザードが起こった場合のストール規則を記述する。データ・ハザード・ストール規則はパイプライン動作中にデータ・ハザードが起こった場合のストール規則を記述する。なお、ストール規則については後で詳しく説明する。

【0016】（動作モデル生成部7）動作モデル生成部7はプロセッサ仕様3からプロセッサの動作モデル5を生成する。生成された動作モデル5は検証項目集合生成部11、検証項目集合編集部15および検証プログラム生成部19で使用される。図4に動作モデル生成部7の動作を示すフローチャートを示す。動作モデル生成部7の処理は、（1）パイプライン構造グラフの作成（ステップ41）、（2）構造ハザードによるストール条件の決定（ステップ42）、（3）データ・ハザードによるストール条件の決定（ステップ43）、（4）有限状態機械（finite state machine）の生成（ステップ44）、に分かれる。

【0017】（1）パイプライン構造グラフの作成
まず、動作モデル生成部7はパイプライン構造グラフを作成する。パイプライン構造グラフはパイプラインを構成する各ステージの接続関係を表す有向グラフGである。Gはパイプライン動作仕様に記されているパイプライン・ステージを節点、パイプライン・ステージ間の接続を有向辺とし、同一のパイプライン・ステージを表す節点を一つにまとめることにより作成される。図5に図3に示したパイプライン動作仕様から得られるパイプライン構造グラフを示す。

【0018】（2）構造ハザードによるストール条件の決定
次に、動作モデル生成部7は構造ハザードによるストール条件を決定する。ステップ1で求めたパイプライン構造グラフの節点のうち複数の有向辺の終点となる節点が

構造ハザードの原因となる。たとえば図5に示したパイプライン構造グラフでは、3個の有向辺の終点となる節点「WB」が構造ハザードの原因となる。構造ハザードによるストール条件は、まず構造ハザード状態集合を求め、この構造ハザード状態集合と構造ハザードストール規則を用いて構造ハザードによるストール条件を決定する。

【0019】構造ハザード状態集合は、パイプライン構造グラフから2個以上の有向辺の終点となる節点の集合 S を求め、集合 S のすべての要素 s について次の処理を行なうことで求められる。まず、要素 s を終点に持つ有向辺の集合 E を求める。そして、集合 E の要素 e のすべての対 (e_i, e_j) （但し、 (e_i, e_j) と (e_j, e_i) は同一視する）について、 e_i の始点を s_i 、 e_j の始点を s_j とし、有効グラフ G の節点 n に対応するパイプラインステージに於ける可能性のある命令の集合を $Inst(n)$ とする。 $Inst(s_i)$ のすべての要素 ik および $Inst(s_j)$ のすべての要素 jl について $((s_i, ik), (s_j, jl))$ を構造ハザード状態集合の要素に加える。ステージが異なっても、同一のリソースを使用する場合、構造ハザードが起こる。このような場合も、ほぼ同様の方法を用いることにより、構造ハザード状態を求めることが可能である。図6に図3に示したパイプライン動作仕様に対する構造ハザード状態集合を示す。

【0020】構造ハザード状態集合が得られたら、構造ハザード状態集合のすべての要素についてハザードが起きた時の動作を決定し、すべての構造ハザード状態集合の要素に対応するパイプラインの状態に対して構造ハザードが起こった時のストール条件を決定する。この時に構造ハザード・ストール規則を用いる。図7に構造ハザード・ストール規則の例を示す。この例では、命令間に優先順位を決めておき優先順位の最も高いもの以外の命令がストールするという規則を表している。例えば、

「 $((ALU, int), (FPU, fpu)) - (ALU, Int)$ 」は、 ALU ステージに int 命令が存在し、 FPU ステージに fpu 命令が存在する場合、 ALU ステージに存在する int 命令がストールすることを表している。図8に図7の規則から得られた構造ハザードによるストール条件を示す。たとえば図8の第2列は、 ALU ステージに存在する int 命令が構造ハザードによりストールする条件は“MEMステージに ld 命令が存在するか FPU ステージに fpu 命令が存在する時である”ことを表している。また第3列は、MEMステージに存在する ld 命令は構造ハザードによりストールすることはないことを表している。

【0021】図9に構造ハザード・ストール規則の他の例を示す。この例では、構造ハザードを引き起こす命令の命令発行順序を比較し、発行順が最も早い命令以外の命令がストールするという規則を表している。 PS_i 、

PS_j をパイプライン・ステージとしたとき、 $Order(PS_i) > Order(PS_j)$ が真とは、 PS_i に存在する命令が PS_j に存在する命令より後に発行されたことを表わすとする。図10に図9の規則から得られた構造ハザードによるストール条件を示す。

【0022】なお、本発明の第1の実施の形態において、決定した構造ハザードによるストール条件の編集（変更）機能を有していることが好ましい。構造ハザード・ストール規則では表現できないものや構造ハザード・ストール規則の例外等を扱うことができるようにするためである。

【0023】(3) データ・ハザードによるストール条件の決定

次に、動作モデル生成部7はデータ・ハザードによるストール条件を決定する。まず、命令セット・アーキテクチャ仕様からデータ・ハザードが起こる命令対の集合（以下、データ・ハザード状態集合と呼ぶ）を求める。図3のパイプライン動作仕様ではFETステージでレジスタ・フェッチが行なわれるため、データ・ハザードのチェックはFETステージにおいて行なう。データ・ハザード状態集合の要素を $(Inst(FET), (PS, Inst(PS)))$ とする。データ・ハザードは先行命令のデスティネーション・レジスタと後続命令のソース・レジスタが一致する場合に起こる。したがって、図2の命令セット・アーキテクチャ仕様および図3のパイプライン動作仕様の場合、データ・ハザード状態集合は図11に示すものとなる。

【0024】データ・ハザード状態集合が得られたら、データ・ハザード状態集合のすべての要素についてハザードが起きた時の動作を決定し、各命令対に対してデータ・ハザードによるストール条件を決める。この時にデータ・ハザード・ストール規則を用いる。図12にデータ・ハザード・ストール規則の例を示す。この例では、○印がついている命令対でデータ・ハザードが起こった場合、FETステージに存在する命令がストールすることを表している。また、図13にデータ・ハザード・ストール規則の他の例を示す。この例では、パイプライン・ステージの前半でレジスタへの書き込みを行ない、パイプライン・ステージの後半でレジスタの参照を行なうという実装にすることで、WBステージで書き込まれた値を同一サイクルのFETステージで参照可能となり、WBステージに存在する命令との間でデータ・ハザードによるストールが起こらなくなる場合を示している。さらに図14にデータ・ハザード・ストール規則の他の例を示す。この例は、フォワーディングを行なうことにより「 ld 」命令以外の命令でデータ・ハザードによるストールが起こらなくなる場合である。

【0025】図15に図12の規則から得られたデータ・ハザード・ストール条件を示す。図15において、「 $Inst(PS)$ 」はパイプライン・ステージ PS に

現在存在する命令を返す関数である。「DHStall Rule」はFETステージに存在する命令、FET以外のステージおよびそのステージに存在する命令から、FETステージに存在する命令がデータ・ハザードによりストールする可能性があるか否かを返す関数であり、図12～図14に示したデータ・ハザード・ストール規則で対応するものが○印の場合“1”を返し、×印の場合“0”を返す。「src」は命令のソース・レジスタの集合を返す関数であり、「dest」は命令のデスティネーション・レジスタを返す関数である。

【0026】なお、本発明の第1の実施の形態において、決定したデータ・ハザードによるストール条件の編集（変更）機能を有していることが好ましい。データ・ハザード・ストール規則では表現できないものやデータ・ハザード・ストール規則の例外等を扱うことができるようにするためである。

【0027】（4）有限状態機械（finite state machine）の生成

次に、動作モデル生成部7は各パイプライン・ステージの命令の組み合わせを状態とみなした有限状態機械を生成する。以下では説明簡略化のため、構造ハザードによるストール条件については図7に示した構造ハザード・ストール規則を用いて決定するものとする。一方、データ・ハザードによるストールについてはここでは考えないものとする。以降では、後段のパイプライン・ステージがストールしたらストールするというストール条件を「暗黙のストール条件」と呼ぶことにする。パイプライン・ステージPSのストール条件を「Stall (PS)」、パイプライン・ステージPSの暗黙のストール条件を「ImplicitStall (PS)」、パイプライン・ステージPSの構造ハザード・ストール条件を「SHStall (PS)」、パイプライン・ステージPSのデータ・ハザード・ストール条件を「DHStall (PS)」で表すことにする。

【0028】図16に「Stall (PS)」を表す式を示す。また、図17に「ImplicitStall (PS)」を表す式を示す。但し、図17において、PSSetは全パイプライン・ステージからなる集合である。また、「Next (PS, I)」はパイプライン・ステージPSに存在する命令がIである場合の次段のパイプライン・ステージを返す関数である。図18は図3のパイプライン動作仕様に対する関数Nextを表で示したものである。また、図19は論理式で示したものである。これらはパイプライン動作仕様から自動的に生成することができる。

【0029】パイプライン・ステージPSに存在する命令がIである時の構造ハザード・ストール条件を「SHStall (PS, I)」とすれば、「SHStall (PS)」は図20で示される式で表される。またパイプライン・ステージPSに存在する命令がIである時の

データ・ハザード・ストール条件を「DHStall (PS, I)」とすれば、「DHStall (PS)」は図21で示される式で表される。但し、図20、図21において、InstSetは全命令からなる集合である。図7に示した構造ハザード・ストール規則から導かれる「SHStall (PS, I)」を図22に示す。この「SHStall (PS, I)」は図4のステップ2で求めた構造ハザード・ストール条件である。図20に示した「SHStall (PS)」の式を用いれば、「SHStall (PS)」は図23に示すものとなる。一方、データ・ハザードによるストールについては考慮しないので、「DHStall (PS)」は図24に示すものとなる。

【0030】パイプライン構造グラフGの節点間に以下に示す関係を定義することにより節点集合は半順序集合となる。

【0031】定義1：パイプライン構造グラフGを有向グラフとし、eをGの有向辺とする。mがeの始点、nがeの終点のとき $m > n$ である。

【0032】「ImplicitStall (PS)」および「Stall (PS)」を定義1の順序に従って作成すれば、すなわち「ImplicitStall (PS)」および「Stall (PS)」作成時にPSより小さいパイプライン・ステージPSjの「ImplicitStall (PSj)」および「Stall (PSj)」が作成済みであるようにすれば、「ImplicitStall (PS)」および「Stall (PS)」を求めることができる。たとえば図3に示したパイプライン動作仕様の場合は、WB→MEM→FPU→ALU→FETの手順で求めることができる。その結果を、図25、図26、図27に示す。

【0033】パイプライン・ステージPSの次状態関数「NextInst (PS)」を求めるにはパイプライン・ステージPSに対応する節点を以下の3つのケースに分けて考える。

【0034】ケース1：節点を終点としてもつ有向辺が存在しない場合

ケース2：節点を始点としてもつ有向辺が存在しない場合

ケース3：上記以外の場合

有向グラフGが1個の節点のみからなる場合にはケース1かつケース2という状況が起こり得るが、この場合グラフはパイプラインを表しているとはいえない。したがって、パイプラインを考える場合は必ず3つのケースは互いに独立となる。ケース1の場合「NextInst (PS)」は図28に示す式、ケース2の場合「NextInst (PS)」は図29に示す式、ケース3の場合「NextInst (PS)」は図30に示す式となる。但し、図18において、「Inst (IN)」は次サイクルに入力される命令を表わすものとし、図29、

図30において、「Bubble」はステージに命令が存在しないことを表わすものとする。さらに、図29、図30において、「PSSet」は全パイプライン・ステージからなる集合であるとする。また、図31に図2の命令セット・アーキテクチャ仕様および図3にパイプライン動作仕様に対して実際に求めた次状態関数を示す。

【0035】このようにして動作モデル生成部7は命令セット・アーキテクチャ仕様、パイプライン動作仕様、構造ハザード・ストール規則およびデータ・ハザード・ストール規則を含むプロセッサ仕様3から動作モデル5（本実施の形態では有限状態機械である）を自動的に生成する。したがって、プロセッサの仕様3に変更があっても、プロセッサ仕様3を修正するだけで、変更されたプロセッサ仕様3に対する動作モデル5を自動的に生成することが可能となる。また、パイプラインのストール動作を変更したい場合でも、ストール規則を修正することにより、変更されたストール動作に対する動作モデルを自動的に生成できる。

【0036】（検証項目集合生成部11）次に、検証項目集合生成部11の動作について説明する。検証項目は各パイプライン・ステージの状態の組み合わせで表現する。具体的にはたとえば図32に示した通りである。検証項目集合生成部11は構造ハザードが起こる各パイプライン・ステージの状態、データ・ハザードが起こる各パイプライン・ステージの状態を列挙する。なお、本実施の形態では、動作モデル生成部7がストール条件を生成する際にストールが起こるパイプラインの状態の組み合わせを求めている。したがって、検証項目集合生成部11は動作モデル生成部7に含まれることになる。

【0037】（検証項目集合編集部15）次に、検証項目集合編集部15の動作について説明する。検証項目集合編集部15は検証項目生成部11で作成された検証項目集合9を編集する。検証項目集合編集部15の主な目的は既存の検証プログラム13が存在すれば、既存の検証プログラム13で検証可能な検証項目を検証項目集合9から取り除くことである。検証項目集合9を編集することにより検証プログラム生成部19において不要な検証項目についての検証プログラムの生成を抑制することができる。

【0038】検証項目集合編集部15は既存の検証プログラム13および動作モデル5に基づいて検証項目集合9を編集する。検証項目集合編集部15は既存の検証プログラム13を動作モデル5に与え、簡易シミュレーションを行なう。動作モデル5ではパイプラインの制御動作のみをモデル化しているため、オペランドで与えられるデータがどのように加工されるかはシミュレートしない。したがって、通常のシミュレーションと比べて高速に実行できる。各サイクル毎に各パイプライン・ステージの状態を記録し、その状態が検証項目集合9に含まれ

る場合には検証項目集合9からその状態を取り除く。

【0039】（検証プログラム生成部19）次に、検証プログラム生成部19の動作について説明する。検証プログラム生成部19は動作モデル5および検証項目集合9から検証項目集合9の各要素を検証するための命令列（検証プログラム）17を自動生成する。検証項目は各パイプライン・ステージに存在する命令により表現されている。また動作モデル5は各パイプライン・ステージに存在する命令を状態とみなした有限状態機械として表現されている。これらを用いた命令列の自動生成は、BDDを用いることにより初期状態から到達可能な状態集合を計算することで、容易に実現可能である。

【0040】本発明の第1の実施の形態によれば、プロセッサ仕様から自動的に動作モデルを生成できる。それにより動作モデル作成にかかる時間を短縮することができる。また、プロセッサの仕様に変更があった場合、命令セット・アーキテクチャ仕様もしくはパイプライン仕様を修正することにより、変更された仕様に対する動作モデルを自動的に生成することが可能となる。またパイプラインのストール動作を変更したい場合でも、ストール規則を修正することにより、変更されたストール動作に対する動作モデルを自動的に生成することが可能となる。

【0041】また、本発明の第1の実施の形態によれば、従来人手により作成していた検証項目をプロセッサ仕様から自動的に生成することができる。このことにより検証項目の洩れを防ぐことができる。また人手作成に比べて、検証項目の作成に要する時間を減らすことができる。この効果は回路の複雑さが増せば増すほど顕著に表れる。

【0042】さらに、本発明の第1の実施の形態によれば、既存の検証プログラムにより検証される検証項目を仕様から作成した検証項目集合から取り除くことができる。このことにより、不要な検証プログラムの生成を抑制することができる。ランダムなテストプログラム自動生成をアルゴリズム的なテストプログラム自動生成を組み合わせる行なう時にも有効である。

【0043】（第2の実施の形態）次に、本発明の第2の実施の形態について説明する。図33は、本発明の第2の実施の形態は、第1の実施の形態同様、パイプライン化したプロセッサを設計する場合に本発明を適用した例である。図33に示すように、本発明の第2の実施の形態に係るプロパティ自動生成装置21は、プロセッサ仕様23を入力し、プロセッサ仕様23を解析する仕様解析部25と、その解析結果を入力し、動作モデルを生成する動作モデル生成部27と、プロセッサの回路記述29を入力し、プロセッサの回路記述29を読み込む回路記述読み込み部31と、その読み込み結果を入力し、プロセッサの回路記述29を解析する記述解析部33と、動作モデル生成部27が生成した動作モデルと記述

解析部 33 の解析結果を入力し、プロセッサのプロパティ 35 を生成するプロパティ生成部 37 とを備えて構成される。

【0044】プロセッサの仕様 23 には、図 1 に示した第 1 の実施の形態のプロセッサ仕様 3 と同様、設計対象のプロセッサの命令セット・アーキテクチャ (ISA) 仕様、各命令のパイプライン動作仕様、パイプラインのストール規則が含まれる。仕様解析部 25 はプロセッサ仕様 23 を解析し、動作モデル生成部 27 が必要とする情報を提供する。仕様解析部 25 が提供する情報に基づいて動作モデル生成部 27 は動作モデルを生成する。本実施の形態では、仕様解析部 25 が最初にプロセッサ仕様 23 を解析しているが、第 1 の実施の形態のように動作モデル生成部 27 が直接プロセッサ仕様 23 を入力する構成としてもよい。

【0045】プロセッサの回路記述 29 はたとえばレジスタ転送レベル (RTL) で記述される。回路記述読み込み部 31 は回路記述 29 を読み込み、記述解析部 33 がプロパティ生成部 37 が必要とする情報を回路記述 29 から取り出す。プロパティ生成部 37 は動作モデル生成部 27 が生成した動作モデルと記述解析部 33 が提供する回路記述についての情報からプロパティ 35 を自動生成する。

【0046】次に、図 33 に示した動作モデル生成部 27 の動作モデル生成とプロパティ生成部 37 のプロパティ生成について説明する。図 34 は、図 33 の動作モデル生成部 27 の動作モデル生成とプロパティ生成部 37 のプロパティ生成の処理手順を示すフローチャートである。まず、動作モデル生成 (ステップ 342) は図 1 に示した本発明の第 1 の実施の形態の動作モデル生成部 7 の動作と同様であるのでここでは説明しない。また、ハザード状態生成 (ステップ 341) は構造ハザードが起る各パイプライン・ステージの状態、データ・ハザードが起る各パイプライン・ステージの状態を列挙する。本第 2 の実施の形態ではストール条件を生成する際、ストールが起るパイプラインの状態の組み合わせを求めているため、動作モデル生成 (ステップ 342) でハザード状態を生成している。ハザード状態は各パイプライン・ステージの状態の組み合わせで表現される。具体的には図 32 に示した通りである。なお、検証対象動作 39 はたとえばパイプラインのハザードである。

【0047】次に、ハザード解消状態生成 (ステップ 343) の処理について説明する。ハザード状態生成 (ステップ 341) で構造ハザード状態集合およびデータ・ハザード状態集合が求められている。また、動作モデル生成 (ステップ 342) で各パイプライン・ステージの命令の組み合わせを一つの状態とみなした有限状態機械が生成されている。したがって、ハザード状態と有限状態機械の次状態関数「NextInst」を用いてハザード解消状態を求めることができる。具体的には、構造

ハザード状態集合 SHSS とデータ・ハザード状態集合 DHSS の各状態 s に対して、まず各パイプライン・ステージ PSi に対して $NextInst(PSi)$ を計算し、ハザード解消状態 Hs に各 $NextInst(PSi)$ を要素として加えることで求めることができる。図 35 に、その例に対応したハザード解消状態を図 36 に示す。

【0048】次に、レジスタ名とシグナル名のマッチング (ステップ 344) の処理について説明する。レジスタ名とシグナル名のマッチング (ステップ 344) ではハザード状態生成 (ステップ 341) およびハザード解消状態生成 (ステップ 343) で生成された状態、すなわち各パイプライン・ステージの命令とプロセッサの RTL 記述 41 内のレジスタおよびシグナルとの対応付けを行う。例えば、パイプライン・ステージ FET に命令 int が存在している場合、RTL 記述 41 中ではレジスタ $Fet_reg[3:0]$ に値 0001 が格納されている状態がパイプライン・ステージ FET に命令 int が存在する場合に対応するとき、「 $Inst(FET) == int$ 」と「 $Fet_reg[3:0] = "0001"$ 」の対応付けを行う。

【0049】次に、プロパティ生成 (ステップ 345) の処理について説明する。例えば、「ハザード状態 s にあるとき、次のクロック・サイクルでハザード解消状態 s' になる」という型のプロパティを生成する場合を考える。図 35 および図 36 に示した例で説明する。各パイプライン・ステージの命令と RTL 記述中のレジスタとの対応関係の例を図 37 に示す。図 37 には図 35 および図 36 に示した例で必要な部分のみを示した。プロパティ生成 (ステップ 345) は、ハザード状態生成 (ステップ 341) およびハザード解消状態生成 (ステップ 343) で生成された状態、すなわち各パイプライン・ステージの命令の組み合わせを、レジスタ名とシグナル名マッチング (ステップ 344) で対応付けられた各パイプライン・ステージの命令と RTL 記述 41 中のレジスタの対応関係を用いて、RTL 記述 41 中のレジスタに置き換えることによりプロパティ 43 を生成する。図 35 および図 36 に示した例に対するプロパティ 43 を図 38 に示す。

【0050】本発明の第 2 の実施の形態によれば、大規模プロセッサの動作検証の際に必要な検証対象動作を表すプロパティをプロセッサ仕様とプロセッサの回路記述から自動的に作成することができる。それにより短期間で動作検証のためのプロパティを用意することが可能となる。

【0051】(第 3 の実施の形態) 次に、本発明の第 3 の実施の形態について説明する。本発明の第 3 の実施の形態は、第 2 の実施の形態で示した動作モデル生成とプロパティ生成の他の例を示すものである。本発明の第 3 の実施の形態は、バス調停を設計する場合に本発明を適

用した例である。図39に本第3の実施の形態に用いるプロセッサ仕様23の例を示す。この例はバス調停のアルゴリズムがラウンド・ロビンであり、バスに接続されるコンポーネント（クライアントと呼ぶ）を示している。図40に本第3の実施の形態に用いる検証対象動作39の例を示す。ここでは、図39および図40に示した例に対してプロパティを自動生成する場合について説明する。図41は、本発明の第3の実施の形態の動作を示すフローチャートである。

【0052】図39に示したラウンド・ロビン型のバス調停とは、バス使用要求を出したクライアントを待ち行列に置かれた順に一定時間ずつ処理し、クロック・サイクル毎に待ち行列の先頭にあったものを最後尾に置く調停方式である。まず、動作モデル生成（ステップ411）でラウンド・ロビン型バス調停を実現する上で必要なクライアント数、各クライアントの優先順位を生成する。その例を図42に示す。この例では、クライアント数は8であり、クライアントの優先順位は適当に生成した。

【0053】次に、レジスタ名とシグナル名のマッチング（ステップ412）では、動作モデル生成（ステップ411）で生成したモデルの各クライアントのバス使用要求、使用許可とRTL記述41内のレジスタ名との対応関係を生成する。RTL記述41の例を図43に、クライアント名とレジスタ名の対応関係の表を図44に示す。

【0054】最後に、プロパティ生成（ステップ413）では、「クライアントC_{L0}がバスをリクエストするとき、8クロック・サイクル以内に許可される」という型のプロパティを生成すると仮定すれば、レジスタ名、シグナル名マッチングの処理で得られた対応関係の表を用いてクライアント名をRTL記述中のレジスタ名に置き換え、プロパティ43を生成する。生成されたプロパティの例を図45に示す。

【0055】本発明の第3の実施の形態においても第2の実施の形態と同様の効果を得ることができる。

【0056】

【発明の効果】本発明によれば、プロセッサの動作検証を短時間でこなうことが可能となる。したがって、プロセッサの設計時間の短縮を図ることができる。

【図面の簡単な説明】

【図1】本発明の第1の実施の形態に係る検証プログラム自動生成装置の構成を示すブロック図である。

【図2】命令セット・アーキテクチャ仕様の例を示す図である。

【図3】パイプライン動作仕様の例を示す図である。

【図4】図1の動作モデル生成部の動作を示すフローチャートである。

【図5】図3のパイプライン動作仕様から得られるパイプライン構造グラフを示す図である。

【図6】図3のパイプライン動作仕様に対する構造ハザード状態集合を示す図である。

【図7】構造ハザード・ストール規則の例を示す図である。

【図8】図7の規則から得られた構造ハザードによるストール条件を示す図である。

【図9】構造ハザード・ストール規則の他の例を示す図である。

【図10】図9の規則から得られた構造ハザードによるストール条件を示す図である。

【図11】図2の命令セット・アーキテクチャ仕様および図3のパイプライン動作仕様に対するデータ・ハザード状態集合を示す図である。

【図12】データ・ハザード・ストール規則の例を示す図である。

【図13】データ・ハザード・ストール規則の他の例を示す図である。

【図14】データ・ハザード・ストール規則の他の例を示す図である。

【図15】図12の規則から得られたデータ・ハザード・ストール条件を示す図である。

【図16】パイプライン・ステージPSのストール条件「Stall (PS)」を表す式を示す図である。

【図17】パイプライン・ステージPSの暗黙のストール条件「Implicit Stall (PS)」を表す式を示す図である。

【図18】図3のパイプライン動作仕様に対する関数Nextを表で示した図である。

【図19】図3のパイプライン動作仕様に対する関数Nextを論理式で示した図である。

【図20】パイプライン・ステージの構造ハザード・ストール条件を表す式を示す図である。

【図21】パイプライン・ステージのデータ・ハザード・ストール条件を表す式を示す図である。

【図22】各パイプライン・ステージの全命令に対する構造ハザード・ストール条件を表す表を示す図である。

【図23】各パイプライン・ステージの構造ハザード・ストール条件を表す表を示す図である。

【図24】各パイプライン・ステージに対応するデータ・ハザード・ストール条件を表す表を示す図である。

【図25】各パイプライン・ステージの全命令に対する暗黙のストール条件を表す表を示す図である。

【図26】各パイプライン・ステージに対する暗黙のストール条件を表す表を示す図である。

【図27】各パイプライン・ステージに対するストール条件を表す表を示す図である。

【図28】前段のパイプライン・ステージが存在しないパイプライン・ステージに対する次サイクルの命令を表す式を示す図である。

【図29】前段および後段のパイプライン・ステージが

存在するパイプライン・ステージに対する次サイクルの命令を表す式を示す図である。

【図30】後段のパイプライン・ステージが存在しないパイプライン・ステージに対する次サイクルの命令を表す式を示す図である。

【図31】各パイプライン・ステージに対する次サイクルの命令を表す表を示す図である。

【図32】図2の命令セット・アーキテクチャ仕様および図3のパイプライン動作仕様で規定される動作モデルを有限状態機械で表現する時に用いる状態を示す図である。

【図33】本発明の第2の実施の形態に係るプロパティ自動生成装置の構成を示すブロック図である。

【図34】、図33の動作モデル生成部の動作モデル生成とプロパティ生成部のプロパティ生成の処理手順を示すフローチャートである。

【図35】ハザード状態の例を示す図である。

【図36】ハザード解消状態の例を示す図である。

【図37】各パイプライン・ステージの命令とRTL記述中のレジスタとの対応関係の例を示す図である。

【図38】図35および図36に示した例に対するプロパティを示す図である。

【図39】本発明の第3の実施の形態に用いるプロセッサ仕様の例を示す図である。

【図40】本発明の第3の実施の形態に用いる検証対象動作の例を示す図である。

【図41】本発明の第3の実施の形態の動作を示すフロ

ーチャートである。

【図42】動作モデルの例を示す図である。

【図43】RTL記述の例を示す図である。

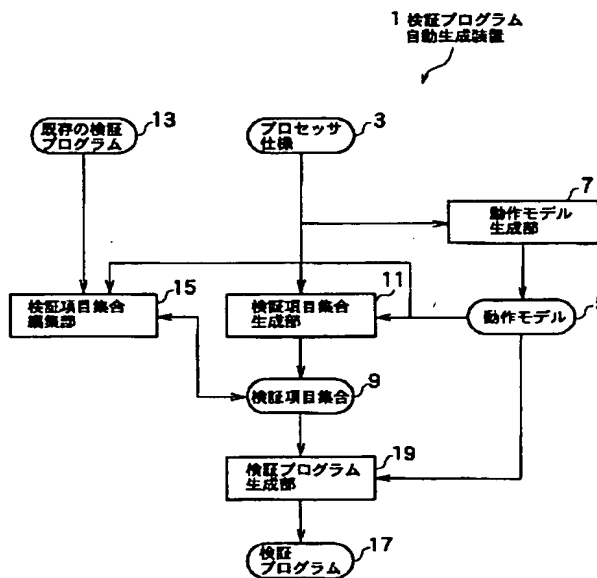
【図44】クライアント名とレジスタ名の対応関係の表を示す図である。

【図45】生成されたプロパティを示す図である。

【符号の説明】

- 1 検証プログラム自動生成装置
- 3、23 プロセッサ仕様
- 5 動作モデル
- 7、27 動作モデル生成部
- 9 検証項目集合
- 11 検証項目集合生成部
- 13 既存の検証プログラム
- 15 検証項目集合編集部
- 17 検証プログラム
- 19 検証プログラム生成部
- 21 プロパティ自動生成装置
- 25 仕様解析部
- 29 プロセッサの回路記述
- 31 回路記述読み込み部
- 33 記述解析部
- 35 プロパティ
- 37 プロパティ生成部
- 39 検証対象動作
- 41 RTL記述
- 43 プロパティ

【図1】



【図2】

命令	dest reg	src reg	命令動作
nop	—	—	—
addi	rd	rs, rt	rd ← rs + rt
sub	rd	rs, rt	rd ← rs - rt
fadd	rd	rs, rt	rd ← rs + rt
fsub	rd	rs, rt	rd ← rs - rt
ld	rd	base, offset	rd ← memory[base + offset]

【図3】

【図9】

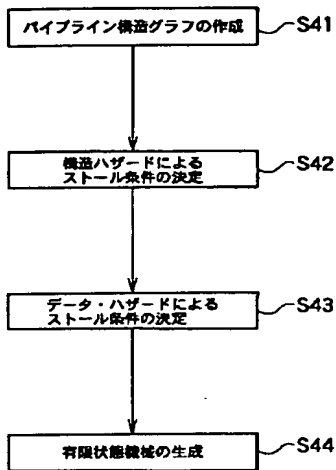
命令	パイプライン動作
nop	FET
int	FET → ALU → WB
fpu	FET → FPU → WB
ld	FET → ALU → MEM → WB

Order(PSi) > Order(PSj) → PSi

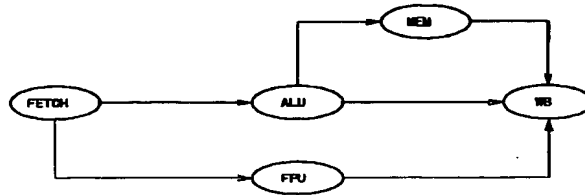
【図6】

構造ハザード状態集合 = [((ALU, int), (FPU, fpu))
 ((ALU, int), (MEM, ld))
 ((FPU, fpu), (MEM, ld))]

【図4】



【図5】



【図18】

PS	Inst	Next(PS, Inst)
FET	int	ALU
	fpu	FPU
ALU	int	WB
	ld	MEM
FPU	fpu	WB
MEM	ld	WB

【図8】

パイプライン・ステージの状態	構造ハザード・ストール条件
(ALU,int)	Inst(MEM) = ld Inst(FPU) = fpu
(MEM,ld)	0
(FPU,fpu)	Inst(MEM) = ld

【図24】

PS	DHStall(PS)
WB	0
MEM	0
FPU	0
ALU	0
FET	0

【図7】

((ALU,int),(FPU,fpu)) → (ALU,int)
 ((ALU,int),(MEM,ld)) → (ALU,int)
 ((FPU,fpu),(MEM,ld)) → (FPU,fpu)

DHStallRule(Inst(FET),(PS,Inst(PS))) & (src(Inst(FET)) ∩ dest(Inst(PS)))

【図15】

【図10】

パイプライン・ステージの状態	構造ハザード・ストール条件
(ALU,int)	Order(Inst(ALU)) < Order(Inst(MEM)) Order(Inst(ALU)) < Order(Inst(FPU))
(MEM,ld)	Order(Inst(MEM)) < Order(Inst(ALU)) Order(Inst(MEM)) < Order(Inst(FPU))
(FPU,fpu)	Order(Inst(FPU)) < Order(Inst(ALU)) Order(Inst(FPU)) < Order(Inst(MEM))

【図11】

データ・ハザード集合 = {

(int,(ALU,int)),(int,(ALU,ld)),
 (int,(FPU,fpu)),(int,(MEM,ld)),
 (int,(WB,int)),(int,(WB,fpu)),(int,(WB,ld)),

(fpu,(ALU,int)),(fpu,(ALU,ld)),

(fpu,(FPU,fpu)),(fpu,(MEM,ld)),

(fpu,(WB,int)),(fpu,(WB,fpu)),(fpu,(WB,ld)),

(ld,(ALU,int)),(ld,(ALU,ld)),

(ld,(FPU,fpu)),(ld,(MEM,ld)),

(ld,(WB,int)),(ld,(WB,fpu)),(ld,(WB,ld))

}

【図12】

		ALU		FPU	MEM	WB		
		int	ld	fpu	ld	int	fpu	ld
FET	nop	x	x	x	x	x	x	x
	int	○	○	○	○	○	○	○
	fpu	○	○	○	○	○	○	○
	ld	○	○	○	○	○	○	○

【図13】

		ALU		FPU	MEM	WB		
		int	ld	fpu	ld	int	fpu	ld
FET	nop	x	x	x	x	x	x	x
	int	○	○	○	○	x	x	x
	fpu	○	○	○	○	x	x	x
	ld	○	○	○	○	x	x	x

【図16】

Stal(PS) = SHStal(PS) | DHStal(PS) | ImplicitStal(PS)

【図 14】

		ALU		FPU	MEM	WB		
		int	ld	fpu	ld	int	fpu	ld
FET	nop	x	x	x	x	x	x	x
	int	x	x	x	○	x	x	x
	fpu	x	x	x	○	x	x	x
	ld	x	x	x	○	x	x	x

【図 17】

$$\text{ImplicitStall}(PS) = \begin{cases} 0 & (n=0) \\ \sum_{NPS \in \text{PSSet}} (\text{Next}(PS, \text{Inst}(PS)) = NPS) \& \text{Stall}(\text{Inst}(NPS)) & (n \geq 1) \end{cases}$$

【図 21】

【図 19】

$\text{Next}(PS, \text{Inst}) = ((PS = \text{FET}) \& (\text{Inst}(PS) = \text{int}) \& \text{ALU})$
 $\vee ((PS = \text{FET}) \& (\text{Inst}(PS) = \text{fpu}) \& \text{FPU})$
 $\vee ((PS = \text{FET}) \& (\text{Inst}(PS) = \text{ld}) \& \text{ALU})$
 $\vee ((PS = \text{ALU}) \& (\text{Inst}(PS) = \text{int}) \& \text{WB})$
 $\vee ((PS = \text{ALU}) \& (\text{Inst}(PS) = \text{ld}) \& \text{MEM})$
 $\vee ((PS = \text{FPU}) \& (\text{Inst}(PS) = \text{fpu}) \& \text{WB})$
 $\vee ((PS = \text{MEM}) \& (\text{Inst}(PS) = \text{ld}) \& \text{WB})$

【図 20】

$$\text{SHStall}(PS) = \sum_{I \in \text{InstSet}} \text{SHStall}(PS, I) \& (\text{Inst}(PS) = I)$$

$$\text{DHStall}(PS) = \sum_{I \in \text{InstSet}} \text{DHStall}(PS, I) \& (\text{Inst}(PS) = I)$$

【図 35】

【図 23】

(int, int, fpu, -, int)

【図 22】

PS	Inst	SHStall(PS, I)
WB	int	0
	fpu	0
	ld	0
MEM	ld	0
FPU	fpu	$\text{Inst}(\text{MEM}) = \text{ld}$
ALU	int	$\text{Inst}(\text{MEM}) = \text{ld} \vee \text{Inst}(\text{FPU}) = \text{fpu}$
	ld	0
	ld	0
FET	int	0
	fpu	0
	ld	0

PS	SHStall(PS)
WB	0
MEM	0
FPU	$\text{Inst}(\text{MEM}) = \text{ld} \& \text{Inst}(\text{FPU}) = \text{fpu}$
ALU	$(\text{Inst}(\text{FPU}) = \text{fpu}) \vee \text{Inst}(\text{MEM}) = \text{ld} \& \text{Inst}(\text{ALU}) = \text{int}$
FET	0

【図 26】

【図 25】

PS	ImplicitStall(PS)
WB	0
MEM	0
FPU	0
ALU	0
FET	$(\text{Inst}(\text{MEM}) = \text{ld} \vee \text{Inst}(\text{FPU}) = \text{fpu}) \& \text{Inst}(\text{ALU}) = \text{int}$ $\& \text{Inst}(\text{FET}) = \text{int} \vee \text{Inst}(\text{FET}) = \text{ld}$ $\vee \text{Inst}(\text{MEM}) = \text{ld} \& \text{Inst}(\text{FPU}) = \text{fpu} \& \text{Inst}(\text{FET}) = \text{fpu}$

【図 27】

PS	Inst	ImplicitStall(PS, Inst)
WB	int	0
	fpu	0
	ld	0
MEM	ld	0
FPU	fpu	0
ALU	int	0
	ld	0
	ld	0
FET	int	$(\text{Inst}(\text{MEM}) = \text{ld} \vee \text{Inst}(\text{FPU}) = \text{fpu}) \& \text{Inst}(\text{ALU}) = \text{int}$
	fpu	$\text{Inst}(\text{MEM}) = \text{ld} \& \text{Inst}(\text{FPU}) = \text{fpu}$
	ld	$(\text{Inst}(\text{MEM}) = \text{ld} \vee \text{Inst}(\text{FPU}) = \text{fpu}) \& \text{Inst}(\text{ALU}) = \text{int}$

PS	Stall(PS)
WB	0
MEM	0
FPU	$\text{Inst}(\text{MEM}) = \text{ld} \& \text{Inst}(\text{FPU}) = \text{fpu}$
ALU	$(\text{Inst}(\text{MEM}) = \text{ld} \vee \text{Inst}(\text{FPU}) = \text{fpu}) \& \text{Inst}(\text{ALU}) = \text{int}$
FET	$(\text{Inst}(\text{MEM}) = \text{ld} \vee \text{Inst}(\text{FPU}) = \text{fpu}) \& \text{Inst}(\text{ALU}) = \text{int}$ $\& \text{Inst}(\text{FET}) = \text{int} \vee \text{Inst}(\text{FET}) = \text{ld}$ $\vee \text{Inst}(\text{MEM}) = \text{ld} \& \text{Inst}(\text{FPU}) = \text{fpu} \& \text{Inst}(\text{FET}) = \text{fpu}$

【図 36】

【図 28】

【図 29】

(int, int, -, -, fpu)

$$\text{NextInst}(PS) = \text{Stall}(PS) \& \text{Inst}(PS) \vee \text{Stall}(PS) \& \text{Inst}(IN) \quad \text{NextInst}(PS) = \text{Stall}(PS) \& \text{Inst}(PS)$$

$$\text{Inst}(PS) \& \sum_{PPS \in \text{PSSet}} \text{Stall}(PPS) \& (\text{Next}(PPS, \text{Inst}(PPS)) = PS) \& \text{Inst}(PPS)$$

$$\text{Inst}(PS) \& \prod_{PPS \in \text{PSSet}} \text{Stall}(PPS) \vee (\text{Next}(PPS, \text{Inst}(PPS)) \neq PS) \& \text{Bubble}$$

【図30】

$$\text{NextInst}(PS) = \text{Inst}(PS) \& \sum_{PPS \in PPS_{st}} \text{Stall}(PPS) \& (\text{Next}(PPS, \text{Inst}(PPS)) = PS) \& \text{Inst}(PPS)$$

$$\text{Inst}(PS) \& \prod_{PPS \in PPS_{st}} \text{Stall}(PPS) \mid (\text{Next}(PPS, \text{Inst}(PPS)) \neq PS) \& \text{Bubble}$$

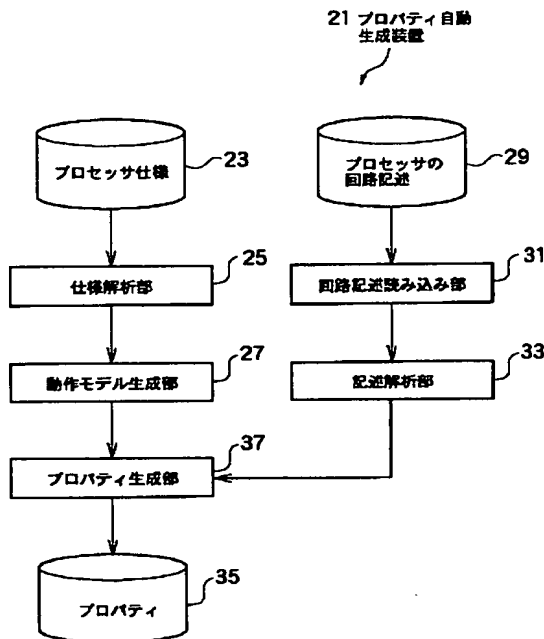
【図32】

(Inst(FET), Inst(ALU), Inst(FPU), Inst(MEM), Inst(WB))

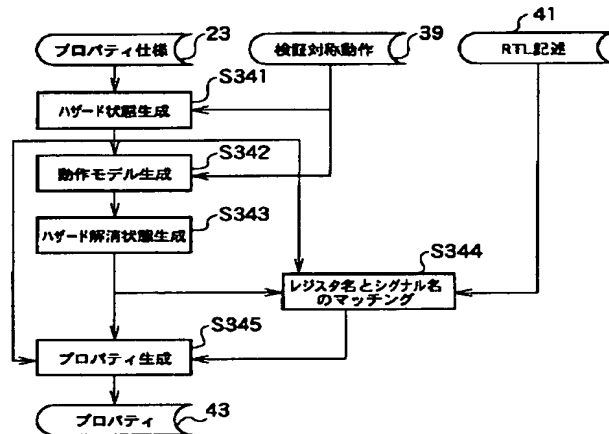
【図31】

PS	NextInst(PS)
FET	Stall(FET)&Inst(FET) Stall(FET)&Inst(IN)
ALU	Stall(ALU)&Inst(ALU) Stall(ALU)&Stall(FET)&(Inst(FET)=INT Inst(FET)=ALU)&Inst(FET) Stall(ALU)&(Stall(FET) Inst(FET)=INT Inst(FET)=ALU)&Bubble
FPU	Stall(FPU)&Inst(FPU) Stall(FPU)&Stall(FET)&Inst(FET)=FPU&Inst(FET) Stall(FPU)&(Stall(FET) Inst(FET)=FPU)&Bubble
MEM	Stall(ALU)&Inst(ALU)=LD&Inst(ALU) Stall(ALU) Inst(ALU)=LD&Bubble
WB	Stall(FPU)&Inst(FPU)=FPU&Inst(FPU) Inst(MEM)=LD&Inst(MEM) (Stall(ALU) Inst(ALU)=INT) & (Stall(FPU) Inst(FPU)=FPU) & Inst(MEM)=LD&Bubble

【図33】



【図34】



【図39】

アルゴリズム	クライアント
ラウンド・ロビン	CL0, CL1, CL2, CL3, CL4, CL5, CL6, CL7

【図38】

【図37】

PS	Inst(PS)	RTL
FET	int	Fet_reg[3:0]="0001"
ALU	int	Alu_reg[3:0]="0001"
FPU	fpu	Fpu_reg[3:0]="0010"
WB	int	Wb_reg[3:0]="0001"
	fpu	Wb_reg[3:0]="0010"

「Fet_reg[3:0]="0001", Alu_reg[3:0]="0001", Fpu_reg[3:0]="0010", Wb_reg[3:0]="0001" があるとき次のクロックサイクルで Fet_reg[3:0]="0001", Alu_reg[3:0]="0001", Wb_reg[3:0]="0010" になる」

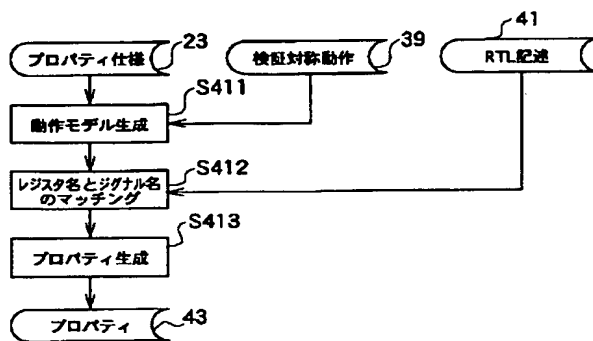
【図42】

クライアント数	8							
クライアント	CL0	CL1	CL2	CL3	CL4	CL5	CL6	CL7
優先順位	1	2	3	4	5	6	7	8

【図40】

クライアント数を8とした時、クライアントがバス要求を出した時、そのクロック・サイクル内に許可される。

【図 4 1】



【図 4 4】

クライアント名	レジスタ名	
	使用要求	使用許可
CL0	request[0]=1	grant[0]=1
CL1	request[1]=1	grant[1]=1
CL2	request[2]=1	grant[2]=1
CL3	request[3]=1	grant[3]=1
CL4	request[4]=1	grant[4]=1
CL5	request[5]=1	grant[5]=1
CL6	request[6]=1	grant[6]=1
CL7	request[7]=1	grant[7]=1

【図 4 3】

```

module round_robin(clk,request,grant)
input clk;
input [ 7:0 ] request;
output [ 7:0 ] grant;
reg [ 7:0 ] grant;
reg [ 7:0 ] slot;

initial slot=0;

always @ (posedge clk)
begin
grant=0;
grant[slot]=request[slot];
if (slot==7)
slot=0;
else
slot=slot+1;
end
endmodule

```

【図 4 5】

「request[0]=1 のとき、8クロック・サイクル内でgrant[0]=1」

フロントページの続き

(72) 発明者 水野 淳

神奈川県川崎市幸区堀川町580番1号 株
式会社東芝半導体システム技術センター内

(72) 発明者 上谷 裕徳

神奈川県川崎市幸区堀川町580番1号 株
式会社東芝半導体システム技術センター内

F ターム (参考) 2G032 AA03 AC08 AE12
5B013 CC01 EE08
5B046 AA08 BA03 JA05
9A001 DD03 HH32 LL08